

Gradient Descent

Nipun Batra

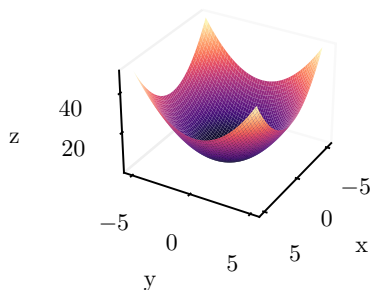
IIT Gandhinagar

August 2, 2025

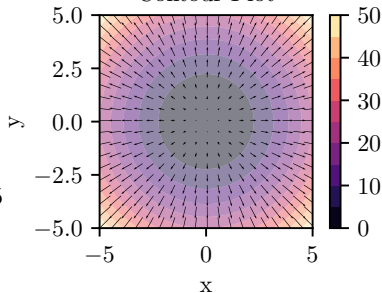
Contour Plot And Gradients

$$z = f(x, y) = x^2 + y^2$$

Surface Plot



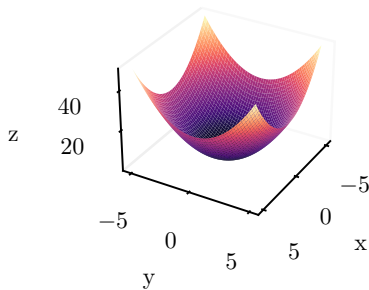
Contour Plot



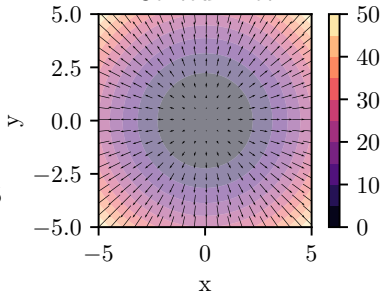
Contour Plot And Gradients

$$z = f(x, y) = x^2 + y^2$$

Surface Plot



Contour Plot

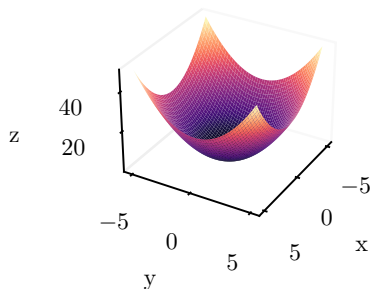


Gradient denotes the direction of steepest ascent or the direction in which there is a maximum increase in $f(x,y)$

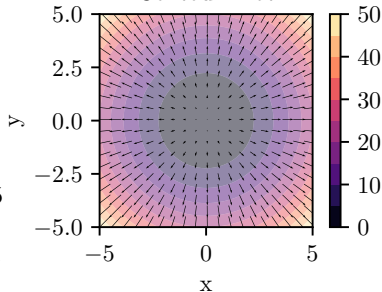
Contour Plot And Gradients

$$z = f(x, y) = x^2 + y^2$$

Surface Plot



Contour Plot



Gradient denotes the direction of steepest ascent or the direction in which there is a maximum increase in $f(x,y)$

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Optimization algorithms

- We often want to minimize/maximize a function

Optimization algorithms

- We often want to minimize/maximize a function
- We wanted to minimize the cost function:

$$f(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \quad (1)$$

Optimization algorithms

- We often want to minimize/maximize a function
- We wanted to minimize the cost function:

$$f(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \quad (1)$$

- Note, here $\boldsymbol{\theta}$ is the parameter vector

Optimization algorithms

- In general, we have following components:

Optimization algorithms

- In general, we have following components:
- Maximize or Minimize a function subject to some constraints

Optimization algorithms

- In general, we have following components:
- Maximize or Minimize a function subject to some constraints
- Today, we will focus on unconstrained optimization (no constraints)

Optimization algorithms

- In general, we have following components:
- Maximize or Minimize a function subject to some constraints
- Today, we will focus on unconstrained optimization (no constraints)
- We will focus on minimization

Optimization algorithms

- In general, we have following components:
- Maximize or Minimize a function subject to some constraints
- Today, we will focus on unconstrained optimization (no constraints)
- We will focus on minimization
- Goal:

$$\theta^* = \arg \min_{\theta} f(\theta) \quad (2)$$

Introduction

- Gradient descent is an optimization algorithm

Introduction

- Gradient descent is an optimization algorithm
- It is used to find the minimum of a function in unconstrained settings

Introduction

- Gradient descent is an optimization algorithm
- It is used to find the minimum of a function in unconstrained settings
- It is an iterative algorithm

Introduction

- Gradient descent is an optimization algorithm
- It is used to find the minimum of a function in unconstrained settings
- It is an iterative algorithm
- It is a first order optimization algorithm

Introduction

- Gradient descent is an optimization algorithm
- It is used to find the minimum of a function in unconstrained settings
- It is an iterative algorithm
- It is a first order optimization algorithm
- It is a local search algorithm/greedy

Gradient Descent Algorithm

1. Initialize θ to some random value

Gradient Descent Algorithm

1. Initialize θ to some random value
2. Compute the gradient of the cost function at θ , $\nabla f(\theta)$

Gradient Descent Algorithm

1. Initialize θ to some random value
2. Compute the gradient of the cost function at θ , $\nabla f(\theta)$
3. For Iteration i ($i = 1, 2, \dots$) or until convergence:

Gradient Descent Algorithm

1. Initialize θ to some random value
2. Compute the gradient of the cost function at θ , $\nabla f(\theta)$
3. For Iteration i ($i = 1, 2, \dots$) or until convergence:
 - $\theta_i \leftarrow \theta_{i-1} - \alpha \nabla f(\theta_{i-1})$

Taylor's Series

- Taylor's series is a way to approximate a function $f(x)$ around a point x_0 using a polynomial

Taylor's Series

- Taylor's series is a way to approximate a function $f(x)$ around a point x_0 using a polynomial
- The polynomial is given by

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (3)$$

Taylor's Series

- Taylor's series is a way to approximate a function $f(x)$ around a point x_0 using a polynomial
- The polynomial is given by

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (3)$$

- The vector form of the above equation is given by:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0) (\mathbf{x} - \mathbf{x}_0) + \dots \quad (4)$$

Taylor's Series

- Taylor's series is a way to approximate a function $f(x)$ around a point x_0 using a polynomial
- The polynomial is given by

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (3)$$

- The vector form of the above equation is given by:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0) (\mathbf{x} - \mathbf{x}_0) + \dots \quad (4)$$

- where $\nabla^2 f(\mathbf{x}_0)$ is the Hessian matrix and $\nabla f(\mathbf{x}_0)$ is the gradient vector

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$
- Then, we have:

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$
- Then, we have:
- $f(x_0) = \cos(0) = 1$

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$
- Then, we have:
- $f(x_0) = \cos(0) = 1$
- $f'(x_0) = -\sin(0) = 0$

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$
- Then, we have:
- $f(x_0) = \cos(0) = 1$
- $f'(x_0) = -\sin(0) = 0$
- $f''(x_0) = -\cos(0) = -1$

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$
- Then, we have:
- $f(x_0) = \cos(0) = 1$
- $f'(x_0) = -\sin(0) = 0$
- $f''(x_0) = -\cos(0) = -1$
- We can write the second order Taylor's series as:

Taylor's Series

- Let us consider $f(x) = \cos(x)$ and $x_0 = 0$
- Then, we have:
- $f(x_0) = \cos(0) = 1$
- $f'(x_0) = -\sin(0) = 0$
- $f''(x_0) = -\cos(0) = -1$
- We can write the second order Taylor's series as:
- $f(x) = 1 + 0(x - 0) + \frac{-1}{2!}(x - 0)^2 = 1 - \frac{x^2}{2}$

Taylor's series

- Let us consider another example: $f(x) = x^2 + 2$ and $x_0 = 2$

Taylor's series

- Let us consider another example: $f(x) = x^2 + 2$ and $x_0 = 2$
- Question: How does the first order Taylor's series approximation look like?

Taylor's series

- Let us consider another example: $f(x) = x^2 + 2$ and $x_0 = 2$
- Question: How does the first order Taylor's series approximation look like?
- First order Taylor's series approximation is given by:

Taylor's series

- Let us consider another example: $f(x) = x^2 + 2$ and $x_0 = 2$
- Question: How does the first order Taylor's series approximation look like?
- First order Taylor's series approximation is given by:
- $f(x) = f(x_0) + f'(x_0)(x - x_0) = 6 + 4(x - 2) = 4x - 2$

Taylor's Series (Alternative form)

- We have:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (5)$$

Taylor's Series (Alternative form)

- We have:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (5)$$

- Let us consider $x = x_0 + \Delta x$ where Δx is a small quantity

Taylor's Series (Alternative form)

- We have:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (5)$$

- Let us consider $x = x_0 + \Delta x$ where Δx is a small quantity
- Then, we have:

$$f(x_0 + \Delta x) = f(x_0) + \frac{f'(x_0)}{1!}\Delta x + \frac{f''(x_0)}{2!}\Delta x^2 + \dots \quad (6)$$

Taylor's Series (Alternative form)

- We have:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (5)$$

- Let us consider $x = x_0 + \Delta x$ where Δx is a small quantity
- Then, we have:

$$f(x_0 + \Delta x) = f(x_0) + \frac{f'(x_0)}{1!}\Delta x + \frac{f''(x_0)}{2!}\Delta x^2 + \dots \quad (6)$$

- Let us assume Δx is small enough such that Δx^2 and higher order terms can be ignored

Taylor's Series (Alternative form)

- We have:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \quad (5)$$

- Let us consider $x = x_0 + \Delta x$ where Δx is a small quantity
- Then, we have:

$$f(x_0 + \Delta x) = f(x_0) + \frac{f'(x_0)}{1!}\Delta x + \frac{f''(x_0)}{2!}\Delta x^2 + \dots \quad (6)$$

- Let us assume Δx is small enough such that Δx^2 and higher order terms can be ignored
- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!}\Delta x$

Taylor's Series to Gradient Descent

- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$

Taylor's Series to Gradient Descent

- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$
- Or, in vector form: $f(\mathbf{x}_0 + \Delta \mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$

Taylor's Series to Gradient Descent

- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$
- Or, in vector form: $f(\mathbf{x}_0 + \Delta \mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- Goal: Find $\Delta \mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta \mathbf{x})$ is minimized

Taylor's Series to Gradient Descent

- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$
- Or, in vector form: $f(\mathbf{x}_0 + \Delta \mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- Goal: Find $\Delta \mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta \mathbf{x})$ is minimized
- This is equivalent to minimizing $f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$

Taylor's Series to Gradient Descent

- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$
- Or, in vector form: $f(\mathbf{x}_0 + \Delta \mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- Goal: Find $\Delta \mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta \mathbf{x})$ is minimized
- This is equivalent to minimizing $f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- This happens when vectors $\nabla f(\mathbf{x}_0)$ and $\Delta \mathbf{x}$ are at phase angle of 180°

Taylor's Series to Gradient Descent

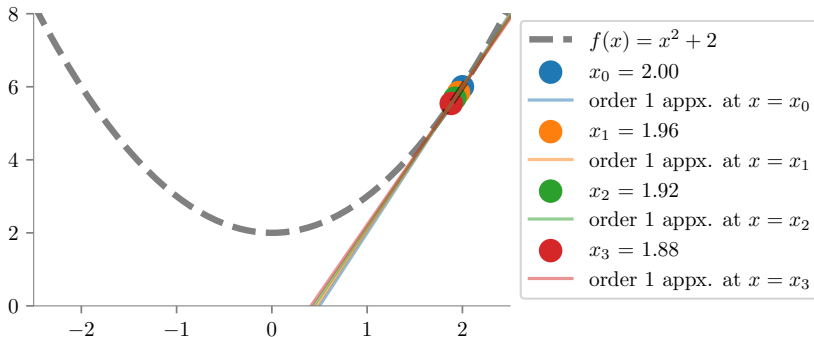
- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$
- Or, in vector form: $f(\mathbf{x}_0 + \Delta \mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- Goal: Find $\Delta \mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta \mathbf{x})$ is minimized
- This is equivalent to minimizing $f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- This happens when vectors $\nabla f(\mathbf{x}_0)$ and $\Delta \mathbf{x}$ are at phase angle of 180°
- This happens when $\Delta \mathbf{x} = -\alpha \nabla f(\mathbf{x}_0)$ where α is a scalar

Taylor's Series to Gradient Descent

- Then, we have: $f(x_0 + \Delta x) \approx f(x_0) + \frac{f'(x_0)}{1!} \Delta x$
- Or, in vector form: $f(\mathbf{x}_0 + \Delta \mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- Goal: Find $\Delta \mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta \mathbf{x})$ is minimized
- This is equivalent to minimizing $f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$
- This happens when vectors $\nabla f(\mathbf{x}_0)$ and $\Delta \mathbf{x}$ are at phase angle of 180°
- This happens when $\Delta \mathbf{x} = -\alpha \nabla f(\mathbf{x}_0)$ where α is a scalar
- This is the gradient descent algorithm:
$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \nabla f(\mathbf{x}_0)$$

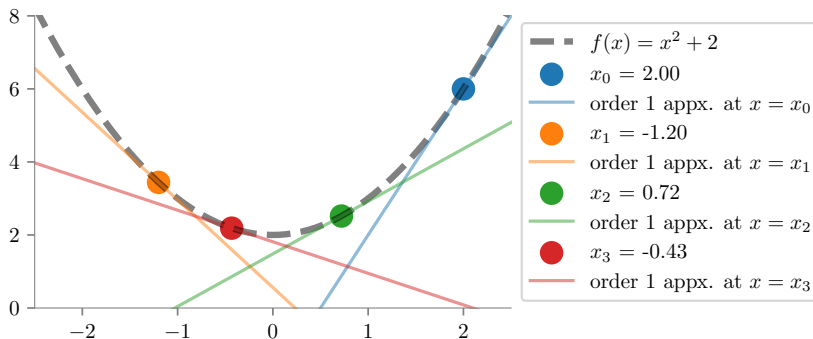
Effect of learning rate

Low learning rate $\alpha = 0.01$: Converges slowly



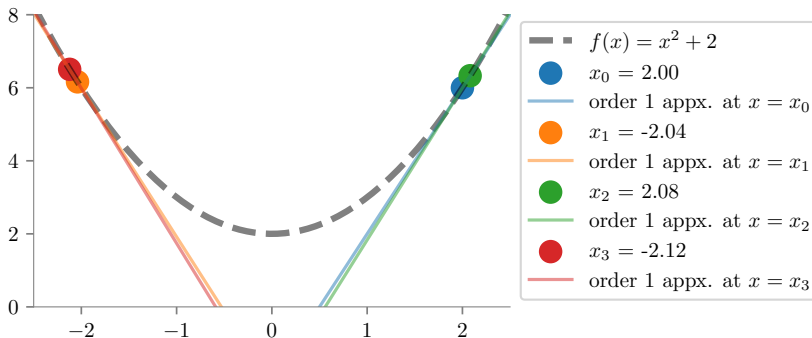
Effect of learning rate

High learning rate $\alpha = 0.8$: Converges quickly, but might overshoot



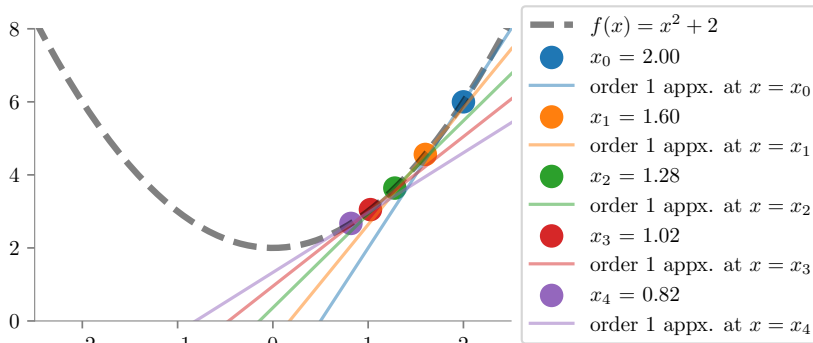
Effect of learning rate

Very high learning rate $\alpha = 1.01$: Diverges



Effect of learning rate

Appropriate learning rate $\alpha = 0.1$



Some commonly confused terms

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.

Some commonly confused terms

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l(f(x_i; \theta), y_i) = (f(x_i; \theta) - y_i)^2$, used in linear regression

Some commonly confused terms

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l(f(x_i; \theta), y_i) = (f(x_i; \theta) - y_i)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:

Some commonly confused terms

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l(f(x_i; \theta), y_i) = (f(x_i; \theta) - y_i)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $\text{MSE}(\theta) = \frac{1}{n} \sum_{i=1}^n (f(x_i; \theta) - y_i)^2$

Some commonly confused terms

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l(f(x_i; \theta), y_i) = (f(x_i; \theta) - y_i)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $\text{MSE}(\theta) = \frac{1}{n} \sum_{i=1}^n (f(x_i; \theta) - y_i)^2$
- **Objective function** is the most general term for any function that you optimize during training.

Gradient Descent : Example

Learn $y = \theta_0 + \theta_1 x$ on following dataset, using gradient descent where initially $(\theta_0, \theta_1) = (4, 0)$ and step-size, $\alpha = 0.1$, for 2 iterations.

x	y
1	1
2	2
3	3

Gradient Descent : Example

Our predictor, $\hat{y} = \theta_0 + \theta_1 x$

Error for i^{th} datapoint, $\epsilon_i = y_i - \hat{y}_i$

$$\epsilon_1 = 1 - \theta_0 - \theta_1$$

$$\epsilon_2 = 2 - \theta_0 - 2\theta_1$$

$$\epsilon_3 = 3 - \theta_0 - 3\theta_1$$

$$\text{MSE} = \frac{\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2}{3} = \frac{14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1}{3}$$

Difference between SSE and MSE

$\sum \epsilon_i^2$ increases as the number of examples increase

So, we use MSE

$$MSE = \frac{1}{n} \sum \epsilon_i^2$$

Here n denotes the number of samples

Gradient Descent : Example

$$\frac{\partial \text{MSE}}{\partial \theta_0} = \frac{2 \sum_{i=1}^n (y_i - \theta_0 - \theta_1 x_i)(-1)}{n} = \frac{2 \sum_{i=1}^n \epsilon_i(-1)}{n}$$

$$\frac{\partial \text{MSE}}{\partial \theta_1} = \frac{2 \sum_{i=1}^n (y_i - \theta_0 - \theta_1 x_i)(-x_i)}{n} = \frac{2 \sum_{i=1}^n \epsilon_i(-x_i)}{n}$$

Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.2 \frac{((1-(4+0))(-1) + (2-(4+0))(-1) + (3-(4+0))(-1))}{3}$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.2 \frac{((1-(4+0))(-1) + (2-(4+0))(-1) + (3-(4+0))(-1))}{3}$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_1 = 0 - 0.2 \frac{((1-(4+0))(-1) + (2-(4+0))(-2) + (3-(4+0))(-3))}{3}$$

$$\theta_1 = -0.67$$

Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.2 \frac{((1 - (3.6 - 0.67))(-1) + (2 - (3.6 - 0.67 \times 2))(-1) + (3 - (3.6 - 0.67 \times 3))(-1))}{3}$$

$$\theta_0 = 3.54$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.2 \frac{((1 - (3.6 - 0.67))(-1) + (2 - (3.6 - 0.67 \times 2))(-1) + (3 - (3.6 - 0.67 \times 3))(-1))}{3}$$

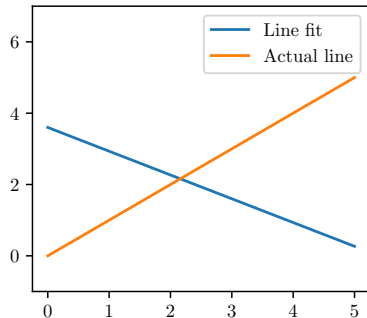
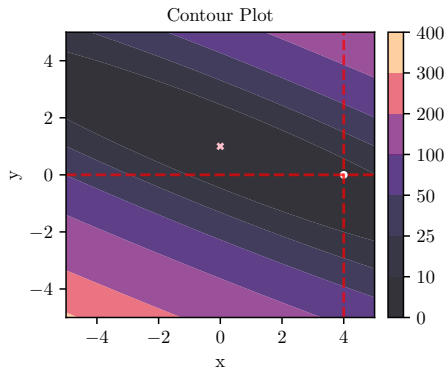
$$\theta_0 = 3.54$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

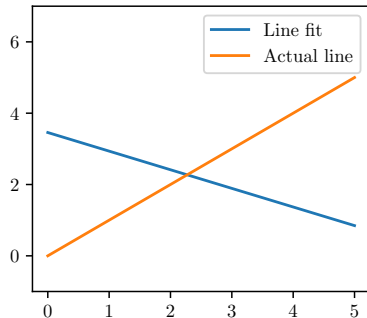
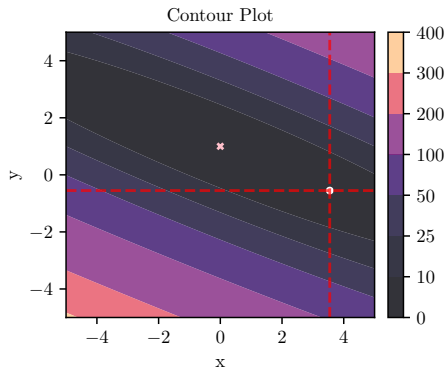
$$\theta_1 = 3.6 - 0.2 \frac{((1 - (3.6 - 0.67))(-1) + (2 - (3.6 - 0.67 \times 2))(-2) + (3 - (3.6 - 0.67 \times 3))(-3))}{3}$$

$$\theta_1 = -0.55$$

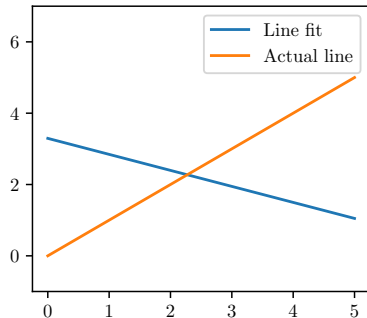
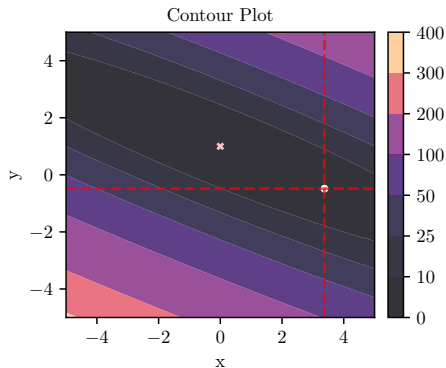
Gradient Descent : Example (Iteration 0)



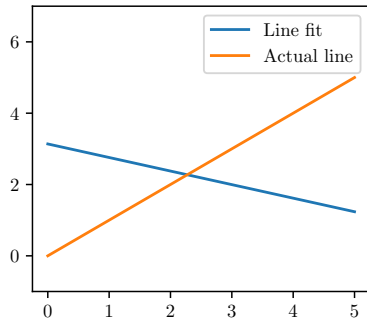
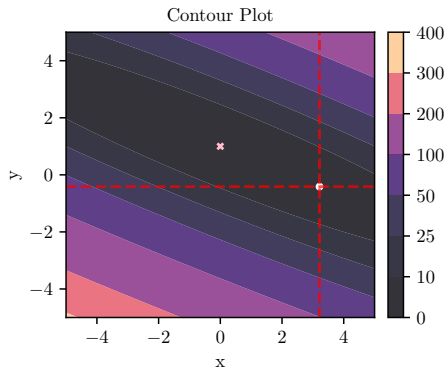
Gradient Descent : Example (Iteration 2)



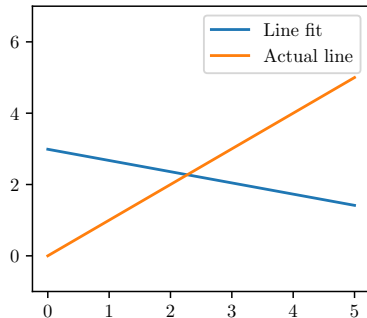
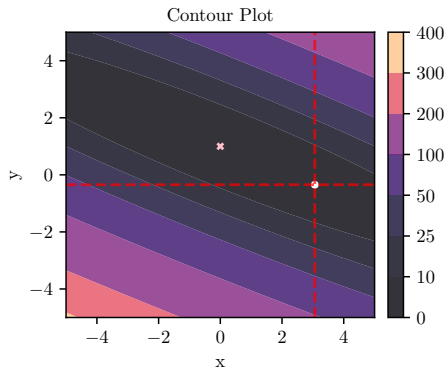
Gradient Descent : Example (Iteration 4)



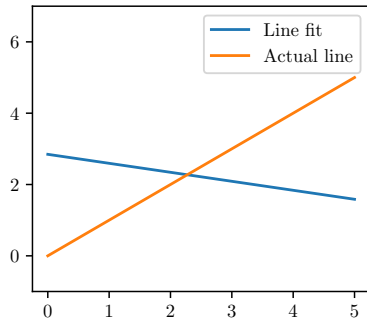
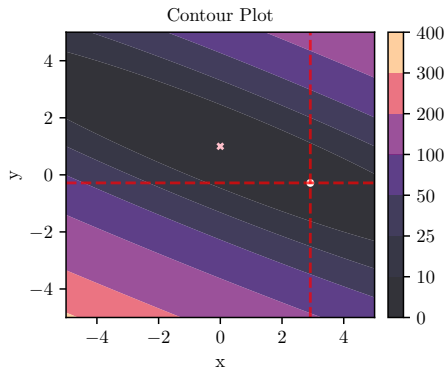
Gradient Descent : Example (Iteration 6)



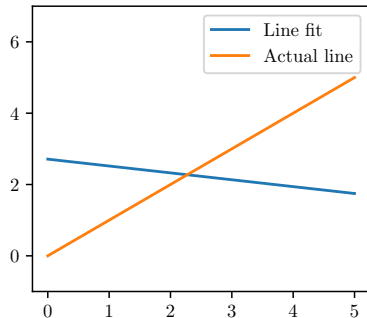
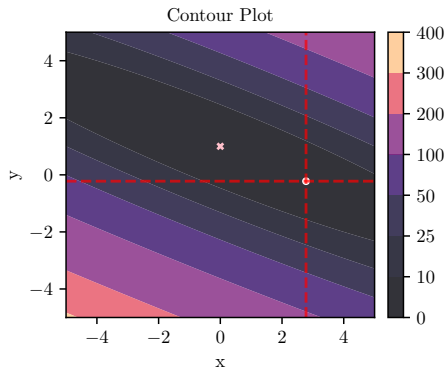
Gradient Descent : Example (Iteration 8)



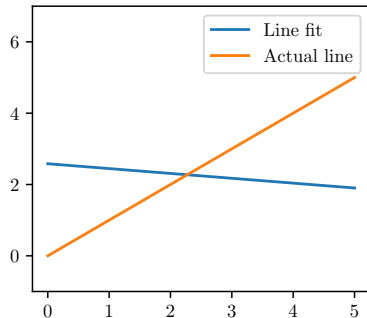
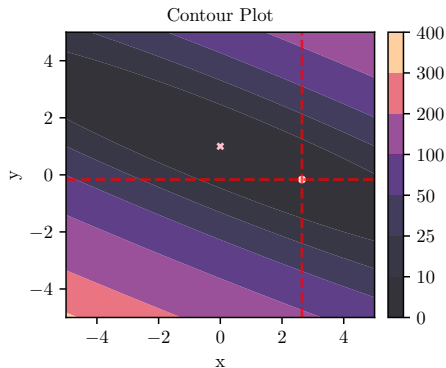
Gradient Descent : Example (Iteraion 10)



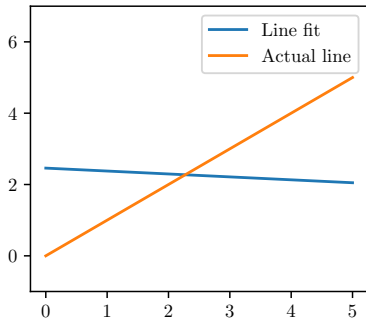
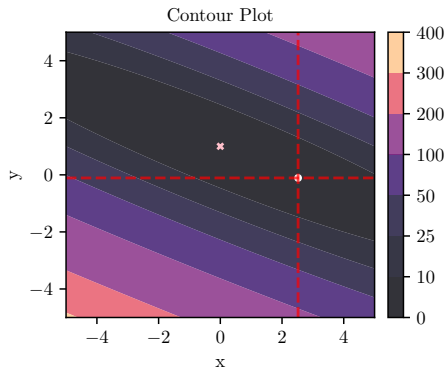
Gradient Descent : Example (Iteration 12)



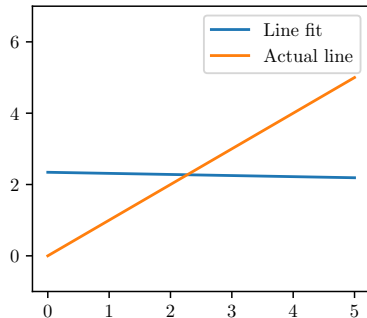
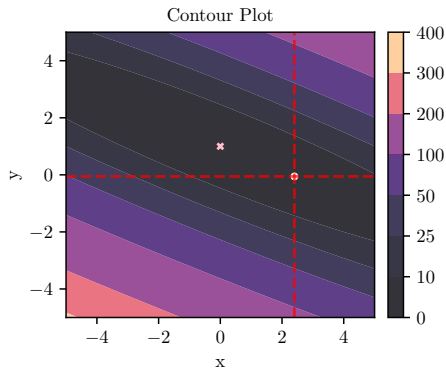
Gradient Descent : Example (Iteration 14)



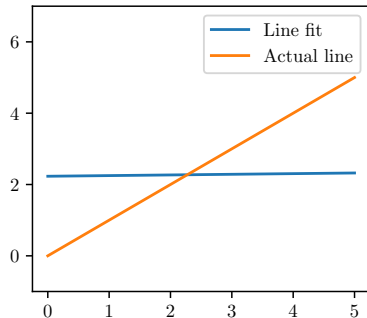
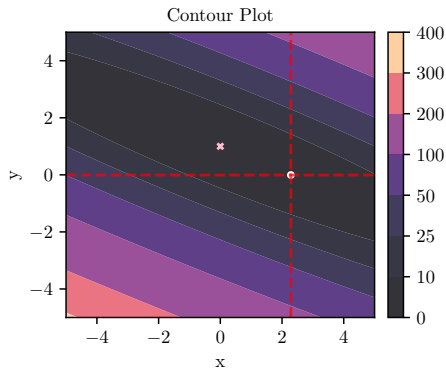
Gradient Descent : Example (Iteration 16)



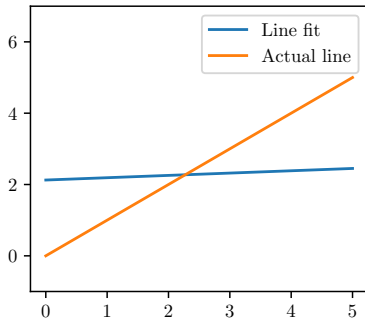
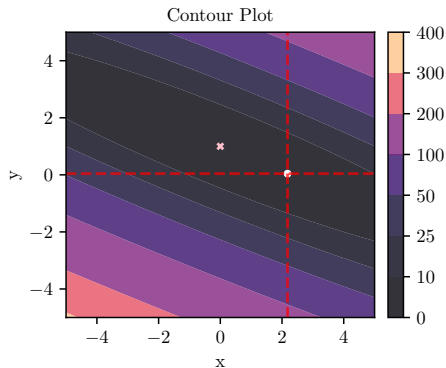
Gradient Descent : Example (Iteraion 18)



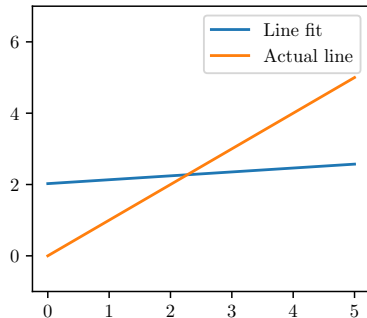
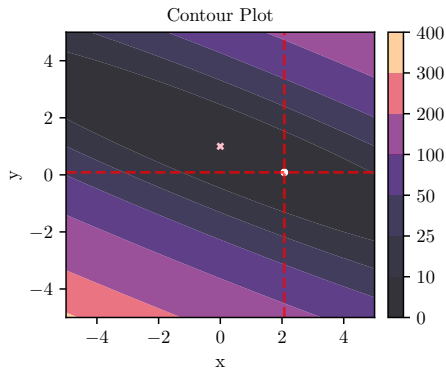
Gradient Descent : Example (Iteration 20)



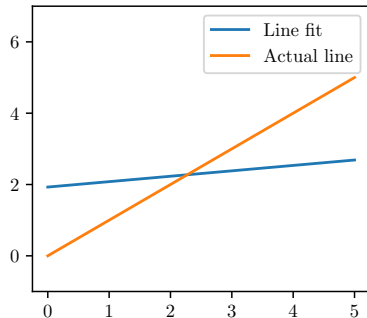
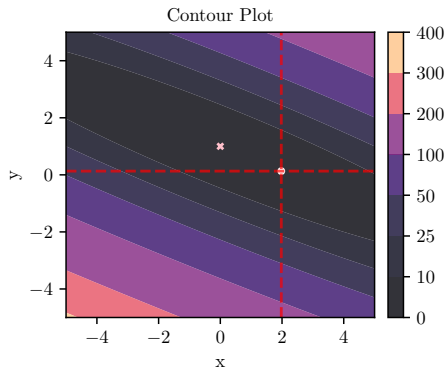
Gradient Descent : Example (Iteration 22)



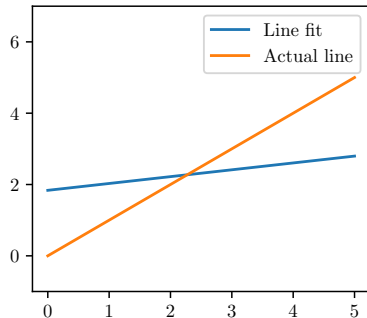
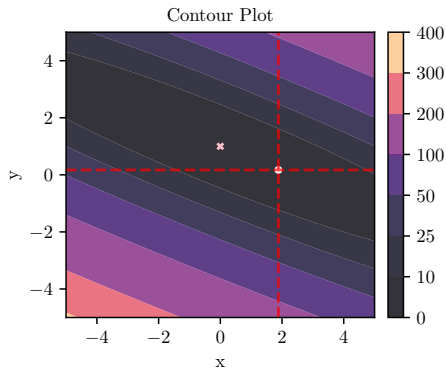
Gradient Descent : Example (Iteration 24)



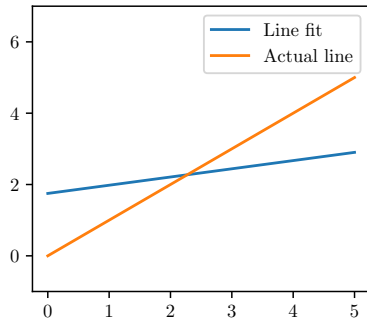
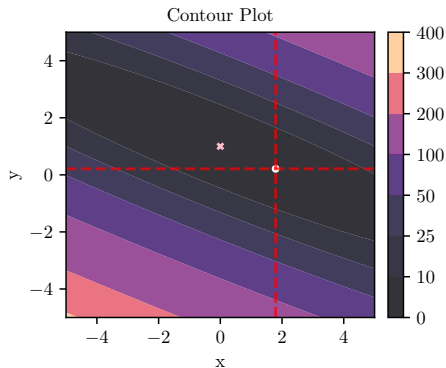
Gradient Descent : Example (Iteration 26)



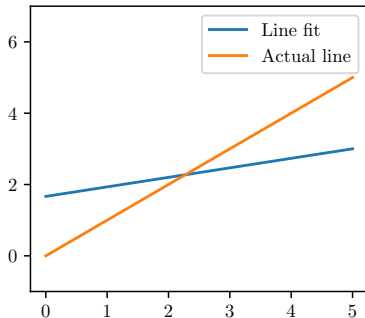
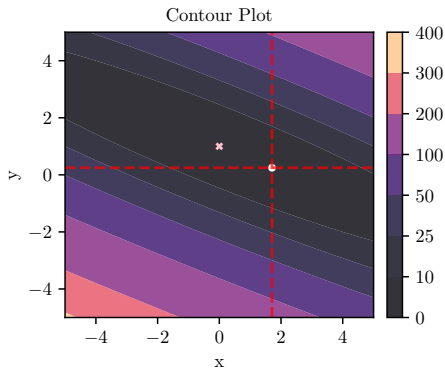
Gradient Descent : Example (Iteration 28)



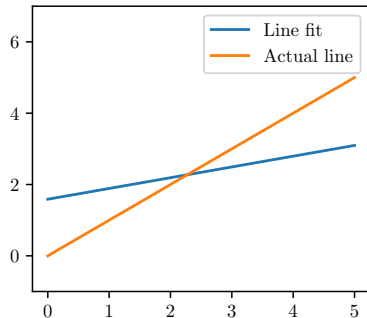
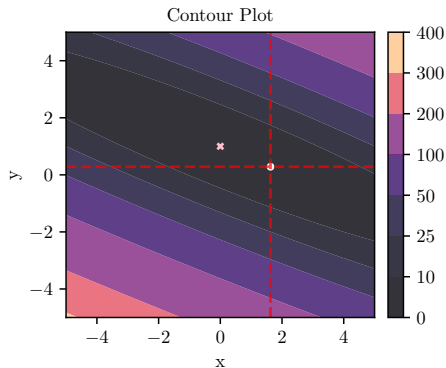
Gradient Descent : Example (Iteration 30)



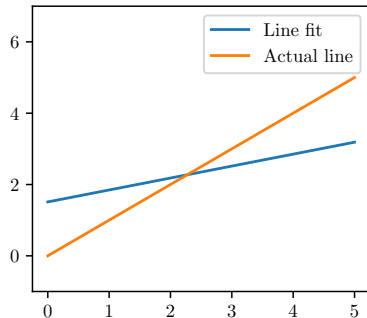
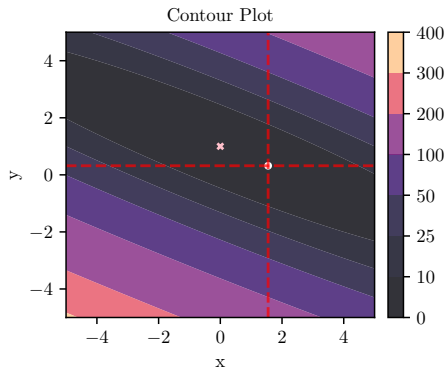
Gradient Descent : Example (Iteration 32)



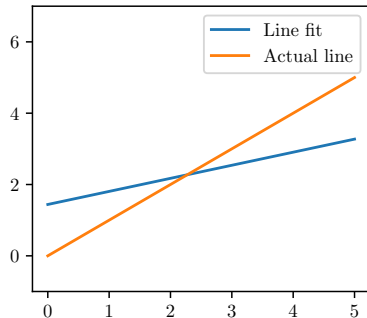
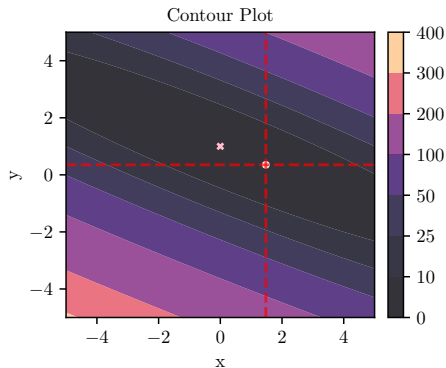
Gradient Descent : Example (Iteration 34)



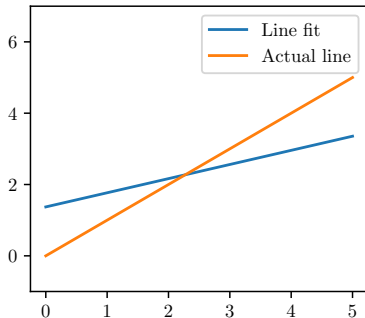
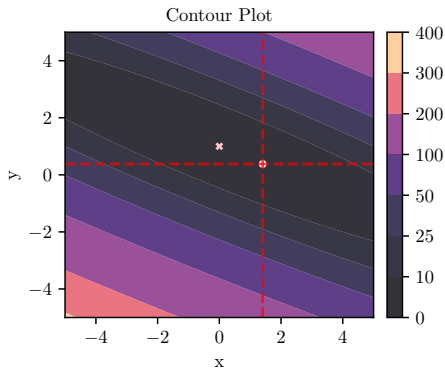
Gradient Descent : Example (Iteration 36)



Gradient Descent : Example (Iteration 38)



Gradient Descent : Example (Iteration 40)



Iteration vs Epochs for gradient descent

- Iteration: Each time you update the parameters of the model

Iteration vs Epochs for gradient descent

- Iteration: Each time you update the parameters of the model
- Epoch: Each time you have seen all the set of examples

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Predict $\hat{\mathbf{y}} = \text{pred}(\mathbf{X}, \theta)$

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Predict $\hat{\mathbf{y}} = \text{pred}(\mathbf{X}, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(\mathbf{y}, \hat{\mathbf{y}})$

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Predict $\hat{\mathbf{y}} = \text{pred}(\mathbf{X}, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(\mathbf{y}, \hat{\mathbf{y}})$
 - Compute gradient: $\nabla J(\theta) = \text{grad}(J)(\theta)$

Gradient Descent (GD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Predict $\hat{\mathbf{y}} = \text{pred}(\mathbf{X}, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(\mathbf{y}, \hat{\mathbf{y}})$
 - Compute gradient: $\nabla J(\theta) = \text{grad}(J)(\theta)$
 - Update: $\theta = \theta - \alpha \nabla J(\theta)$

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - For i in $[1, n]$

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - For i in $[1, n]$
 - Predict $\hat{y}_i = \text{pred}(\mathbf{x}_i, \theta)$

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - For i in $[1, n]$
 - Predict $\hat{y}_i = \text{pred}(\mathbf{x}_i, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(y_i, \hat{y}_i)$

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - For i in $[1, n]$
 - Predict $\hat{y}_i = \text{pred}(\mathbf{x}_i, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(y_i, \hat{y}_i)$
 - Compute gradient: $\nabla J(\theta) = \text{grad}(J)(\theta)$

Stochastic Gradient Descent (SGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - For i in $[1, n]$
 - Predict $\hat{y}_i = \text{pred}(\mathbf{x}_i, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(y_i, \hat{y}_i)$
 - Compute gradient: $\nabla J(\theta) = \text{grad}(J)(\theta)$
 - Update: $\theta = \theta - \alpha \nabla J(\theta)$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = `make_batches(\mathcal{D}, B)`

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = `make_batches(\mathcal{D}, B)`
 - For b in Batches

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = $\text{make_batches}(\mathcal{D}, B)$
 - For b in Batches
 - $\mathbf{X}_b, \mathbf{y}_b = b$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = `make_batches(\mathcal{D}, B)`
 - For b in Batches
 - $\mathbf{X}_b, \mathbf{y}_b = b$
 - Predict $\hat{\mathbf{y}}_b = \text{pred}(\mathbf{X}_b, \theta)$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = `make_batches(\mathcal{D}, B)`
 - For b in Batches
 - $\mathbf{X}_b, \mathbf{y}_b = b$
 - Predict $\hat{\mathbf{y}}_b = \text{pred}(\mathbf{X}_b, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(\mathbf{y}_b, \hat{\mathbf{y}}_b)$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = `make_batches(\mathcal{D}, B)`
 - For b in Batches
 - $\mathbf{X}_b, \mathbf{y}_b = b$
 - Predict $\hat{\mathbf{y}}_b = \text{pred}(\mathbf{X}_b, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(\mathbf{y}_b, \hat{\mathbf{y}}_b)$
 - Compute gradient: $\nabla J(\theta) = \text{grad}(J)(\theta)$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ of size n
- Initialize θ
- For epoch e in $[1, E]$
 - Shuffle \mathcal{D}
 - Batches = `make_batches(\mathcal{D}, B)`
 - For b in Batches
 - $\mathbf{X}_b, \mathbf{y}_b = b$
 - Predict $\hat{\mathbf{y}}_b = \text{pred}(\mathbf{X}_b, \theta)$
 - Compute loss: $J(\theta) = \text{loss}(\mathbf{y}_b, \hat{\mathbf{y}}_b)$
 - Compute gradient: $\nabla J(\theta) = \text{grad}(J)(\theta)$
 - Update: $\theta = \theta - \alpha \nabla J(\theta)$

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Stochastic Gradient Descent

- In SGD, we update parameters after seeing each each point

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Stochastic Gradient Descent

- In SGD, we update parameters after seeing each each point
- Noisier curve for iteration vs cost

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Stochastic Gradient Descent

- In SGD, we update parameters after seeing each each point
- Noisier curve for iteration vs cost
- For a single update, it computes the gradient over one example. Hence lesser time

Stochastic Gradient Descent : Example

Learn $y = \theta_0 + \theta_1 x$ on following dataset, using SGD where initially $(\theta_0, \theta_1) = (4, 0)$ and step-size, $\alpha = 0.1$, for 1 epoch (3 iterations).

x	y
2	2
3	3
1	1

Stochastic Gradient Descent : Example

Our predictor, $\hat{y} = \theta_0 + \theta_1 x$

Error for i^{th} datapoint, $e_i = y_i - \hat{y}_i$

$$\epsilon_1 = 2 - \theta_0 - 2\theta_1$$

$$\epsilon_2 = 3 - \theta_0 - 3\theta_1$$

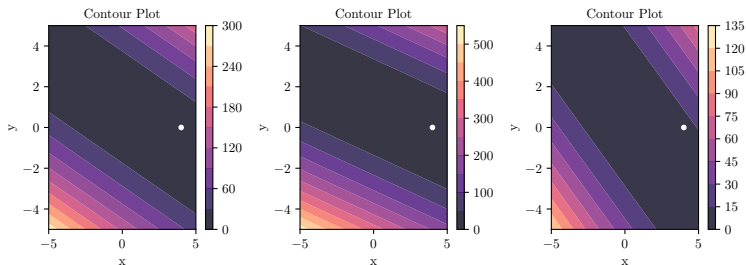
$$\epsilon_3 = 1 - \theta_0 - \theta_1$$

While using SGD, we compute the MSE using only 1 datapoint per iteration.

So MSE is ϵ_1^2 for iteration 1 and ϵ_2^2 for iteration 2.

Stochastic Gradient Descent : Example

Contour plot of the cost functions for the three datapoints



Stochastic Gradient Descent : Example

For Iteration i

$$\frac{\partial MSE}{\partial \theta_0} = 2 (y_i - \theta_0 - \theta_1 x_i) (-1) = 2\epsilon_i (-1)$$

$$\frac{\partial MSE}{\partial \theta_1} = 2 (y_i - \theta_0 - \theta_1 x_i) (-x_i) = 2\epsilon_i (-x_i)$$

Stochastic Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.1 \times 2 \times (2 - (4 + 0)) (-1)$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.1 \times 2 \times (2 - (4 + 0)) (-1)$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_1 = 0 - 0.1 \times 2 \times (2 - (4 + 0)) (-2)$$

$$\theta_1 = -0.8$$

Stochastic Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.1 \times 2 \times (3 - (3.6 - 0.8 \times 3)) (-1)$$

$$\theta_0 = 3.96$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.1 \times 2 \times (3 - (3.6 - 0.8 \times 3)) (-1)$$

$$\theta_0 = 3.96$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_1 = -0.8 - 0.1 \times 2 \times (3 - (3.6 - 0.8 \times 3)) (-3)$$

$$\theta_1 = 0.28$$

Stochastic Gradient Descent : Example

Iteration 3

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 3

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.96 - 0.1 \times 2 \times (1 - (3.96 + 0.28 \times 1)) (-1)$$

$$\theta_0 = 3.312$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 3

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.96 - 0.1 \times 2 \times (1 - (3.96 + 0.28 \times 1)) (-1)$$

$$\theta_0 = 3.312$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_1 = 0.28 - 0.1 \times 2 \times (1 - (3.96 + 0.28 \times 1)) (-1)$$

$$\theta_1 = -0.368$$

True Gradient

Based on Estimation Theory and Machine Learning by
Florian Hartmann

- Let us say we have a dataset \mathcal{D} containing input output pairs $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$

True Gradient

Based on Estimation Theory and Machine Learning by
Florian Hartmann

- Let us say we have a dataset \mathcal{D} containing input output pairs $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- We can define overall loss as:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^N \text{loss}(f(x_i, \theta), y_i)$$

True Gradient

Based on Estimation Theory and Machine Learning by
Florian Hartmann

- Let us say we have a dataset \mathcal{D} containing input output pairs $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- We can define overall loss as:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^N \text{loss}(f(x_i, \theta), y_i)$$

- loss can be any loss function such as squared loss, cross-entropy loss etc.

$$\text{loss}(f(x_i, \theta), y_i) = (f(x_i, \theta) - y_i)^2$$

True Gradient

- The true gradient of the loss function is given by:

$$\begin{aligned}\nabla L &= \nabla \frac{1}{n} \sum_{i=1}^n \text{loss}(f(x_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla \text{loss}(f(x_i), y_i)\end{aligned}$$

True Gradient

- The true gradient of the loss function is given by:

$$\begin{aligned}\nabla L &= \nabla \frac{1}{n} \sum_{i=1}^n \text{loss}(f(x_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla \text{loss}(f(x_i), y_i)\end{aligned}$$

- The above is a consequence of linearity of the gradient operator.

Estimator for the true gradient

- In practice, we do not have access to the true gradient

Estimator for the true gradient

- In practice, we do not have access to the true gradient
- We can only estimate the true gradient using a subset of the data

Estimator for the true gradient

- In practice, we do not have access to the true gradient
- We can only estimate the true gradient using a subset of the data
- For SGD, we use a single example to estimate the true gradient, for mini-batch gradient descent, we use a mini-batch of examples to estimate the true gradient

Estimator for the true gradient

- In practice, we do not have access to the true gradient
- We can only estimate the true gradient using a subset of the data
- For SGD, we use a single example to estimate the true gradient, for mini-batch gradient descent, we use a mini-batch of examples to estimate the true gradient
- Let us say we have a sample: (x, y)

Estimator for the true gradient

- In practice, we do not have access to the true gradient
- We can only estimate the true gradient using a subset of the data
- For SGD, we use a single example to estimate the true gradient, for mini-batch gradient descent, we use a mini-batch of examples to estimate the true gradient
- Let us say we have a sample: (x, y)
- The estimated gradient is given by:

$$\nabla \tilde{L} = \nabla \text{loss}(f(x), y)$$

Bias of the estimator

- One measure for the quality of an estimator \tilde{X} is its bias or how far off its estimate is on average from the true value X :

$$\text{bias}(X) = \mathbb{E}[\tilde{X}] - X$$

Bias of the estimator

- One measure for the quality of an estimator \tilde{X} is its bias or how far off its estimate is on average from the true value X :

$$\text{bias}(X) = \mathbb{E}[\tilde{X}] - X$$

- Using the rules of expectation, we can show that the expected value of the estimated gradient is the true gradient:

$$\begin{aligned}\mathbb{E}[\nabla \tilde{L}] &= \sum_{i=1}^n \frac{1}{n} \nabla \text{loss}(f(\mathbf{x}_i), y_i) \\ &= \frac{1}{n} \nabla \sum_{i=1}^n \text{loss}(f(\mathbf{x}_i), y_i) \\ &= \nabla L\end{aligned}$$

Bias of the estimator

- One measure for the quality of an estimator \tilde{X} is its bias or how far off its estimate is on average from the true value X :

$$\text{bias}(X) = \mathbb{E}[\tilde{X}] - X$$

- Using the rules of expectation, we can show that the expected value of the estimated gradient is the true gradient:

$$\begin{aligned}\mathbb{E}[\nabla \tilde{L}] &= \sum_{i=1}^n \frac{1}{n} \nabla \text{loss}(f(x_i), y_i) \\ &= \frac{1}{n} \nabla \sum_{i=1}^n \text{loss}(f(x_i), y_i) \\ &= \nabla L\end{aligned}$$

- Thus, the estimated gradient is an unbiased estimator of the true gradient

Normal Equation

- Consider $\mathbf{X} \in \mathbb{R}^{n \times d}$

Normal Equation

- Consider $\mathbf{X} \in \mathbb{R}^{n \times d}$
- n examples and d dimensions

Normal Equation

- Consider $\mathbf{X} \in \mathbb{R}^{n \times d}$
- n examples and d dimensions
- What is the time complexity of solving the normal equation $\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$?

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$
- $\mathbf{X}^T \mathbf{X}$ is a matrix product of matrices of size: $d \times n$ and $n \times d$, which is $\mathcal{O}(d^2 n)$

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$
- $\mathbf{X}^T\mathbf{X}$ is a matrix product of matrices of size: $d \times n$ and $n \times d$, which is $\mathcal{O}(d^2n)$
- Inversion of $\mathbf{X}^T\mathbf{X}$ is an inversion of a $d \times d$ matrix, which is $\mathcal{O}(d^3)$

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$
- $\mathbf{X}^T\mathbf{X}$ is a matrix product of matrices of size: $d \times n$ and $n \times d$, which is $\mathcal{O}(d^2n)$
- Inversion of $\mathbf{X}^T\mathbf{X}$ is an inversion of a $d \times d$ matrix, which is $\mathcal{O}(d^3)$
- $\mathbf{X}^T\mathbf{y}$ is a matrix vector product of size $d \times n$ and $n \times 1$, which is $\mathcal{O}(dn)$

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$
- $\mathbf{X}^T\mathbf{X}$ is a matrix product of matrices of size: $d \times n$ and $n \times d$, which is $\mathcal{O}(d^2n)$
- Inversion of $\mathbf{X}^T\mathbf{X}$ is an inversion of a $d \times d$ matrix, which is $\mathcal{O}(d^3)$
- $\mathbf{X}^T\mathbf{y}$ is a matrix vector product of size $d \times n$ and $n \times 1$, which is $\mathcal{O}(dn)$
- $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is a matrix product of a $d \times d$ matrix and $d \times 1$ matrix, which is $\mathcal{O}(d^2)$

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$
- $\mathbf{X}^T\mathbf{X}$ is a matrix product of matrices of size: $d \times n$ and $n \times d$, which is $\mathcal{O}(d^2n)$
- Inversion of $\mathbf{X}^T\mathbf{X}$ is an inversion of a $d \times d$ matrix, which is $\mathcal{O}(d^3)$
- $\mathbf{X}^T\mathbf{y}$ is a matrix vector product of size $d \times n$ and $n \times 1$, which is $\mathcal{O}(dn)$
- $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is a matrix product of a $d \times d$ matrix and $d \times 1$ matrix, which is $\mathcal{O}(d^2)$
- Overall complexity: $\mathcal{O}(d^2n) + \mathcal{O}(d^3) + \mathcal{O}(dn) + \mathcal{O}(d^2) = \mathcal{O}(d^2n) + \mathcal{O}(d^3)$

Normal Equation

- \mathbf{X} has dimensions $n \times d$, \mathbf{X}^T has dimensions $d \times n$
- $\mathbf{X}^T\mathbf{X}$ is a matrix product of matrices of size: $d \times n$ and $n \times d$, which is $\mathcal{O}(d^2n)$
- Inversion of $\mathbf{X}^T\mathbf{X}$ is an inversion of a $d \times d$ matrix, which is $\mathcal{O}(d^3)$
- $\mathbf{X}^T\mathbf{y}$ is a matrix vector product of size $d \times n$ and $n \times 1$, which is $\mathcal{O}(dn)$
- $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is a matrix product of a $d \times d$ matrix and $d \times 1$ matrix, which is $\mathcal{O}(d^2)$
- Overall complexity: $\mathcal{O}(d^2n) + \mathcal{O}(d^3) + \mathcal{O}(dn) + \mathcal{O}(d^2) = \mathcal{O}(d^2n) + \mathcal{O}(d^3)$
- Scales cubic in the number of columns/features of \mathbf{X}

Gradient Descent

Start with random values of θ_0 and θ_1

Till convergence

- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\sum \epsilon_i^2)$

Gradient Descent

Start with random values of θ_0 and θ_1

Till convergence

- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\sum \epsilon_i^2)$
- $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (\sum \epsilon_i^2)$

Gradient Descent

Start with random values of θ_0 and θ_1

Till convergence

- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\sum \epsilon_i^2)$
- $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (\sum \epsilon_i^2)$
- Question: Can you write the above for d dimensional data in vectorised form?

Gradient Descent

Start with random values of θ_0 and θ_1

Till convergence

- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\sum \epsilon_i^2)$
- $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (\sum \epsilon_i^2)$
- Question: Can you write the above for d dimensional data in vectorised form?
- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$
- $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$
- \vdots
- $\theta_d = \theta_d - \alpha \frac{\partial}{\partial \theta_d} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$

Gradient Descent

Start with random values of θ_0 and θ_1

Till convergence

- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\sum \epsilon_i^2)$
- $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (\sum \epsilon_i^2)$
- Question: Can you write the above for d dimensional data in vectorised form?
- $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$
 $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$
 \vdots
 $\theta_d = \theta_d - \alpha \frac{\partial}{\partial \theta_d} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$
- $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$

Gradient Descent

$$\begin{aligned} & \frac{\partial}{\partial \boldsymbol{\theta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \\ &= \frac{\partial}{\partial \boldsymbol{\theta}} (\mathbf{y}^\top - \boldsymbol{\theta}^\top \mathbf{X}^\top) (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \\ &= \frac{\partial}{\partial \boldsymbol{\theta}} (\mathbf{y}^\top \mathbf{y} - \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}) \\ &= -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \\ &= 2\mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \end{aligned}$$

Gradient Descent

We can write the vectorised update equation as follows,
for each iteration

$$\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$$

Gradient Descent

We can write the vectorised update equation as follows, for each iteration

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

For t iterations, what is the computational complexity of our gradient descent solution?

Gradient Descent

We can write the vectorised update equation as follows, for each iteration

$$\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$$

For t iterations, what is the computational complexity of our gradient descent solution?

Hint, rewrite the above as: $\theta = \theta - \alpha \mathbf{X}^\top \mathbf{X} \theta + \alpha \mathbf{X}^\top \mathbf{y}$

Gradient Descent

We can write the vectorised update equation as follows, for each iteration

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

For t iterations, what is the computational complexity of our gradient descent solution?

Hint, rewrite the above as: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} + \alpha \mathbf{X}^\top \mathbf{y}$

Complexity of computing $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(dn)$

Gradient Descent

We can write the vectorised update equation as follows, for each iteration

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

For t iterations, what is the computational complexity of our gradient descent solution?

Hint, rewrite the above as: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} + \alpha \mathbf{X}^\top \mathbf{y}$

Complexity of computing $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(dn)$

Complexity of computing $\alpha \mathbf{X}^\top \mathbf{y}$ once we have $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(d)$ since $\mathbf{X}^\top \mathbf{y}$ has d entries

Gradient Descent

We can write the vectorised update equation as follows, for each iteration

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

For t iterations, what is the computational complexity of our gradient descent solution?

Hint, rewrite the above as: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} + \alpha \mathbf{X}^\top \mathbf{y}$

Complexity of computing $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(dn)$

Complexity of computing $\alpha \mathbf{X}^\top \mathbf{y}$ once we have $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(d)$ since $\mathbf{X}^\top \mathbf{y}$ has d entries

Complexity of computing $\mathbf{X}^\top \mathbf{X}$ is $\mathcal{O}(d^2n)$ and then multiplying with α is $\mathcal{O}(d^2)$

Gradient Descent

We can write the vectorised update equation as follows, for each iteration

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

For t iterations, what is the computational complexity of our gradient descent solution?

Hint, rewrite the above as: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} + \alpha \mathbf{X}^\top \mathbf{y}$

Complexity of computing $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(dn)$

Complexity of computing $\alpha \mathbf{X}^\top \mathbf{y}$ once we have $\mathbf{X}^\top \mathbf{y}$ is $\mathcal{O}(d)$ since $\mathbf{X}^\top \mathbf{y}$ has d entries

Complexity of computing $\mathbf{X}^\top \mathbf{X}$ is $\mathcal{O}(d^2n)$ and then multiplying with α is $\mathcal{O}(d^2)$

All of the above need only be calculated once!

Gradient Descent

Gradient Descent

For each of the t iterations, we now need to first multiply $\alpha \mathbf{X}^\top \mathbf{X}$ with θ which is matrix multiplication of a $d \times d$ matrix with a $d \times 1$, which is $\mathcal{O}(d^2)$

Gradient Descent

For each of the t iterations, we now need to first multiply $\alpha \mathbf{X}^\top \mathbf{X}$ with θ which is matrix multiplication of a $d \times d$ matrix with a $d \times 1$, which is $\mathcal{O}(d^2)$

The remaining subtraction/addition can be done in $\mathcal{O}(d)$ for each iteration.

Gradient Descent

For each of the t iterations, we now need to first multiply $\alpha \mathbf{X}^\top \mathbf{X}$ with θ which is matrix multiplication of a $d \times d$ matrix with a $d \times 1$, which is $\mathcal{O}(d^2)$

The remaining subtraction/addition can be done in $\mathcal{O}(d)$ for each iteration.

What is overall computational complexity?

Gradient Descent

For each of the t iterations, we now need to first multiply $\alpha \mathbf{X}^\top \mathbf{X}$ with θ which is matrix multiplication of a $d \times d$ matrix with a $d \times 1$, which is $\mathcal{O}(d^2)$

The remaining subtraction/addition can be done in $\mathcal{O}(d)$ for each iteration.

What is overall computational complexity?

$$\mathcal{O}(td^2) + \mathcal{O}(d^2n) = \mathcal{O}((t+n)d^2)$$

Gradient Descent (Alternative)

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$
- Computing $\alpha \mathbf{X}^\top$ is $\mathcal{O}(nd)$

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$
- Computing $\alpha \mathbf{X}^\top$ is $\mathcal{O}(nd)$
- Computing $\alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(nd)$

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$
- Computing $\alpha \mathbf{X}^\top$ is $\mathcal{O}(nd)$
- Computing $\alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(nd)$
- Computing $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(n)$

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$
- Computing $\alpha \mathbf{X}^\top$ is $\mathcal{O}(nd)$
- Computing $\alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(nd)$
- Computing $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(n)$

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$
- Computing $\alpha \mathbf{X}^\top$ is $\mathcal{O}(nd)$
- Computing $\alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(nd)$
- Computing $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(n)$

What is overall computational complexity?

Gradient Descent (Alternative)

If we do not rewrite the expression $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$
For each iteration, we have:

- Computing $\mathbf{X}\theta$ is $\mathcal{O}(nd)$
- Computing $\mathbf{X}\theta - \mathbf{y}$ is $\mathcal{O}(n)$
- Computing $\alpha \mathbf{X}^\top$ is $\mathcal{O}(nd)$
- Computing $\alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(nd)$
- Computing $\theta = \theta - \alpha \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$ is $\mathcal{O}(n)$

What is overall computational complexity?
 $\mathcal{O}(ndt)$